Journal of Computational and Applied Mathematics 29 (1990) 111–127
North-Holland

Parallel iteration of high-order Runge–Kutta methods with stepsize control

P.J. VAN DER HOUWEN and B.P. SOMMEIJER
Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Received 6 December 1988
Revised 3 March 1989

Abstract: This paper investigates iterated Runge–Kutta methods of high order designed in such a way that the right-hand side evaluations can be computed in parallel. Using stepsize control based on embedded formulas a highly efficient code is developed. On parallel computers, the 8th-order mode of this code is more efficient than the DOPRI8 implementation of the formulas of Prince and Dormand. The 10th-order mode is about twice as cheap for comparable accuracies.

Keywords: Numerical analysis, Runge–Kutta methods, parallelism.

1. Introduction

Implicit Runge–Kutta (RK) methods for solving the initial-value problem for the system of ordinary differential equations (ODEs)

$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = f(y(t)), \tag{1.1}$$

are seldom used in predictor–corrector (PC) iteration, because RK correctors are much more expensive than linear multistep (LM) correctors. This is due to the increased number of coupled nonlinear algebraic equations. Although RK correctors of order p usually possess smaller error constants than LM correctors of comparable order, an accuracy-computational effort graph will be in favour of PC methods based on LM methods. However, matters are different when parallel computers are used. It is well known that PC iteration, being a form of functional iteration (or Jacobi iteration), allows a high degree of parallelism, because, by partitioning the system of equations into subsystems of equal computational complexity, we can assign to each processor such a subsystem and perform the iteration steps in parallel. The problem is of course the partitioning in subsystems of equal computational complexity. In the case of iterating s-stage RK methods, there is a natural partitioning based on the s subsystems corresponding to the s stages of the RK method. In this way, the computation time involved in applying RK correctors can be reduced a great deal on parallel computers. We shall call these "parallel, iterated" RK methods *PIRK methods*. The idea of iterating an implicit RK method to exploit parallelism goes back to Jackson and Nørsett [10] and also in [9,11,12] such methods have been debated. Before

continuing our discussion on PC iteration, we emphasize that the choice of an implicit RK corrector has nothing to do with the excellent stability characteristics such methods usually possess, since this property is not preserved when the PC approach is followed. Their choice is solely determined by the fact that a high order of accuracy is easily obtained and, particularly, because of the potential parallelism exhibited by these methods. Hence, in the sequel we will assume that the class of ODEs (1.1) is nonstiff and has to be solved with high accuracy demands.

If the predictor is itself an (explicit) RK method, then the PIRK method also belongs to the class of explicit RK methods. In [9] it was proved that explicit RK methods of order p necessarily require at least p *effective* stages, and in [12] the question was posed whether it is always possible to find explicit RK methods of order p using not more than p effective stages, assuming that sufficiently many processors are available (an explicit RK method is said to have p *effective* stages if the computation time required for evaluating all right-hand sides in one step is p times the computation time required by one right-hand side evaluation). This question motivated us to look in the class of PIRK methods for explicit RK methods, the order of which equals the number of effective stages; such methods will be called *optimal* RK methods. We will show that PIRK methods generated by any (not necessarily implicit) s-stage RK corrector of order p do not require more than p effective stages provided that s processors are available. The next question is the least number of processors needed to implement an optimal explicit RK method. For example, in [12] a 5th-order, 6-stage RK method of Butcher which can be implemented on two processors requiring only 5 effective stages is mentioned. This method is clearly an example of an optimal "minimal processor" RK method. So far, we did not succeed in answering the question of least number of necessary processors. Therefore, we have looked for RK methods of which the number of stages is small with respect to their order. It is well known that, within the class of RK methods, those of Gauss–Legendre type require least number of stages to obtain a given order; to be more precise, s-stage Gauss–Legendre methods have order $p = 2s$. Hence, for an "optimal" implementation of these methods, we need only s processors. Furthermore, the stability regions can directly be derived from known results for truncated Taylor series, they allow an extremely simple implementation, and we obtain automatically a sequence of embedded methods of varying order which can be used for stepsize control. PIRK codes of order 8 and 10 using automatic stepsize control are compared with the code DOPRI8 of [5] which is a variable step implementation of the 8th-order explicit RK formula with 7th-order embedded formula of [13]. All codes use the same stepsize strategy. By a number of experiments, the performance of PIRK codes is demonstrated. Both codes are considerably cheaper than DOPRI8 for comparable accuracies. In the Appendix to this paper, we provide a FORTRAN implementation of the PIRK methods. This implementation has the feature that the user can introduce arbitrary RK correctors by means of their Butcher arrays.

Instead of using (one-step explicit) RK predictors one may use LM predictors reducing the number of effective stages. First results based on LM predictors are reported by Lie [11], using a 4th-order, 2-stage Gauss–Legendre corrector and a 3rd-order Hermite extrapolation predictor. With this PC pair, one iteration suffices to obtain a 4th-order PIRK scheme. We shall briefly discuss the use of multistep predictors, in particular for RK correctors of general (nonquadrature) type. Various predictor methods are compared showing that the efficiency of PIRK methods using multistep predictors is higher, but the price to be paid for the increased efficiency is more storage and a less easy implementation.

Finally, the methods proposed in the following sections will be described for *scalar* differential

equations of the form (1.1). Their application, however, is straightforwardly extended to systems of ODEs.

2. Optimal RK methods

Our starting point is the s-stage, implicit, one-step RK method of the form

$$y_{n+1} = y_n + h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}_{n+1}, \tag{2.1a}$$

where $\boldsymbol{r}_{n+1}$ is implicitly defined by

$$\boldsymbol{r}_{n+1} := f(y_n\boldsymbol{e} + hA\boldsymbol{r}_{n+1}). \tag{2.1b}$$

Here, h is the integration step, $\boldsymbol{e}$ is a column vector of dimension s with unit entries, $\boldsymbol{b}$ is an s-dimensional vector and A is an $s \times s$-matrix. Furthermore, we use the convention that for any given vector $\boldsymbol{v} = (v_j)$, $f(\boldsymbol{v})$ denotes the vector with entries $f(v_j)$. By iterating the equation for $\boldsymbol{r}_{n+1}$ m times by simple functional iteration and using the mth iterate as an approximation to $\boldsymbol{r}_{n+1}$, we obtain the method

$$\boldsymbol{r}^{(j)} = f(y_n\boldsymbol{e} + hA\boldsymbol{r}^{(j-1)}), \quad j = 1, \ldots, m, \qquad y_{n+1} = y_n + h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(m)}. \tag{2.2}$$

Since the s components of the vectors $\boldsymbol{r}^{(j)}$ can be computed in parallel, provided that s processors are available, the computational time needed for one iteration of (2.2) is equivalent to the time required to evaluate one right-hand side function on a sequential computer. Hence, the total costs of (2.2) per integration step comprise the calculation of the initial approximation $\boldsymbol{r}^{(0)}$ plus m right-hand side evaluations. In the following, we always assume that we have s processors at our disposal and, speaking about "computational effort per step", we mean the computational time required per step if s processors are available. If the computational effort per step equals the computation time for performing M right-hand side evaluations, then we shall say that the method requires M *effective stages*. Here, and in the sequel, we have assumed that the costs per step are predominated by the time needed to evaluate the derivative function. If this happens to be not the case for a particular ODE, then the overhead, which is sequential in essence, will take a relative large portion of the total costs per step and, consequently, the parallel evaluation of the s (cheap) right-hand side functions will not result in an overall speedup with a factor s.

We shall call the method providing $\boldsymbol{r}^{(0)}$ the *predictor method* and (2.1) the *corrector method* and the resulting parallel, iterated RK method will be briefly called *PIRK method*. It should be observed that in the present case of RK correctors, the predictor and corrector methods do not directly generate approximations to y_{n+1} as is the case in PC methods based on LM methods. However, at any stage of the iteration process we can compute the current approximation to y_{n+1} by means of the formula

$$y^{(j)} := y_n + h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(j)}, \quad j = 0, 1, \ldots. \tag{2.3}$$

Let $\boldsymbol{r}^{(0)}$ be an approximation to $\boldsymbol{r}_{n+1}$ satisfying the condition

$$\boldsymbol{r}^{(0)} = \boldsymbol{r}_{n+1} + \mathrm{O}(h^q), \tag{2.4}$$

resulting in $y^{(0)} = y_{n+1} + \mathrm{O}(h^{q+1})$. Predictor methods satisfying (2.4) will be called *predictor methods of order q*.

Suppose that A and $\boldsymbol{b}^{\mathrm{T}}$ are such that the corrector (2.1) is of order p and let the predictor method be of order $q-1$. Then, it has been proved in [10] that the (global) order of y_{n+1} as defined by (2.2) equals $p^* := \min\{p, q+m\}$. By using the simple predictor method $\boldsymbol{r}^{(0)} := f(y_n)\boldsymbol{e} = \boldsymbol{r}_{n+1} + \mathrm{O}(h)$, i.e., $q=1$, we immediately have as a corollary of this result the next theorem.

Theorem 1. *Let* $\{A, \boldsymbol{b}^{\mathrm{T}}\}$ *define an s-stage RK method (which need not be implicit) of order p. Then the PIRK method defined by*

$$\boldsymbol{r}^{(0)} = f(y_n)\boldsymbol{e}, \quad \boldsymbol{r}^{(j)} = f\big(y_n\boldsymbol{e} + hA\boldsymbol{r}^{(j-1)}\big), \quad j=1,\ldots,m, \quad y_{n+1} = y_n + h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(m)}, \tag{2.5}$$

represents an $(m+1)$*s-stage explicit RK method of order* $p^* := \min\{p, m+1\}$ *requiring* $m+1$ *effective stages.*

Method (2.5) can also be represented by its Butcher array. Defining the s-dimensional vector $\boldsymbol{0}$ and the $s \times s$-matrix O both with zero entries, we obtain

$$\begin{array}{c|ccccc} O & & & & \\ A & O & & & \\ O & A & O & & \\ \vdots & & & & \\ O & \ldots & O & A & O \\ \hline \boldsymbol{0}^{\mathrm{T}} & \ldots & \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{b}^{\mathrm{T}} \end{array}$$

We remark that this Butcher tableau represents a direct translation of (2.5), resulting in $(m+1)s$ stages. However, written in this form, the O-matrix in the first row could be replaced by a scalar zero, since the prediction $\boldsymbol{r}^{(0)}$ has equal components and, consequently, can be produced by one processor. This would lead to an explicit RK method possessing $ms+1$ stages.

Setting $m = p-1$, it follows from Theorem 1 that the question posed by Nørsett and Simonsen [12] can be answered in the affirmative: any pth-order RK method $\{A, \boldsymbol{b}^{\mathrm{T}}\}$ generates an explicit RK method of the form (2.5) of order p requiring only p effective stages. Such explicit RK methods will be called *optimal RK methods.* Of course, within the class (2.5) the number of processors needed for the implementation is dictated by the number of stages s of the generating corrector. For example, the 10th-order, 17-stage RK method of Hairer [4] generates an explicit RK method of the form (2.5) which is also of order 10 if we set $m=9$ and which is optimal in the above sense. However, the implementation of this method requires 17 processors. This suggests the problem of constructing RK methods of order p which are optimal and require least number of processors. The 5th-order, 6-stage RK method of Butcher mentioned in [12] is an example of such a method: it can be implemented on 2 processors requiring only 5 effective stages. From the theory of RK methods based on high-order quadrature methods, such as Gauss–Legendre and Radau methods [5], we can immediately deduce a lower bound for the number of processors needed to implement optimal RK methods of the form (2.5).

Theorem 2. *RK methods of the form* (2.5) *are optimal if* $m = p-1$. *For even p the least number of required processors equals* $\frac{1}{2}p$ *and the generating RK corrector is the pth-order Gauss–Legendre method; for odd p the least number of processors is* $\frac{1}{2}(p+1)$ *and the generating RK corrector is the pth-order Radau method.*

Table 1
Comparison of sequential RK methods and optimal RK methods of the form (2.5)

	p	$\leqslant 4$	5	6	7	8	9	10
Sequential RK	$s_{\min}$	p	6	7	9	11	$\geqslant 12$	$\geqslant 13$
	S	p	6	7	9	11	–	17
Optimal RK	S_{eff}	p	5	6	7	8	9	10
	S_{pr}	–	3	3	4	4	5	5

Thus, optimal RK methods requiring less than $\lfloor \frac{1}{2}(p+1) \rfloor$ processors cannot be found among the methods of the form (2.5). Since (2.5) allows an extremely simple implementation and provides automatically a sequence of embedded formulas which can be used for error estimation (see Section 5) and order variation, we have not looked for methods requiring less than $\lfloor \frac{1}{2}(p+1) \rfloor$ processors.

In order to illustrate the significance of Theorem 2, we make a comparison with explicit RK methods devised for one-processor computers (sequential methods). In Table 1 the minimal number of stages $s_{\min}$ (and therefore the minimal number of right-hand side evaluations) needed to generate such methods of order p are listed. In addition, we list the number of stages S for which these RK methods have actually been constructed (cf. [5, Section 11.6]), and the numbers of effective stages S_{eff} and processors S_{pr} needed by the optimal RK methods of Theorem 2.

Finally, we remark that if the RK corrector is based on quadrature (or collocation) methods, then the initial approximation $\boldsymbol{r}^{(0)}$ can be interpreted as the derivative $f(\boldsymbol{Y}^{(0)})$, where $\boldsymbol{Y}^{(0)}$ is an approximation to $y(t_n\boldsymbol{e} + h\boldsymbol{A}\boldsymbol{e})$. Suppose that the components of $\boldsymbol{Y}^{(0)}$ are computed (in parallel) by using an explicit $(q-1)$-stage RK method of order $q-1$ with stepsizes $h\boldsymbol{A}\boldsymbol{e}$. The resulting PIRK method is still an explicit RK method itself and it is optimal if $m = p - q$ corrections are performed.

3. Multistep predictor methods

Evidently, we can save computing time by using multistep predictor methods. As observed above, such predictors should provide approximations to the derivative values $f(y(t_n\boldsymbol{e} + h\boldsymbol{A}\boldsymbol{e}))$ in the case where the generating RK method $\{A, \boldsymbol{b}^{\mathrm{T}}\}$ is derived from quadrature formulas. Any set of linear multistep methods providing approximations to the components of $y(t_n\boldsymbol{e} + h\boldsymbol{A}\boldsymbol{e})$ serves this purpose.

In this paper we briefly discuss the case of arbitrary RK correctors where we cannot give an easy interpretation for the initial approximation $\boldsymbol{r}^{(0)}$. In such cases, it is possible to construct multistep predictor methods by performing the auxiliary vector recursion

$$\boldsymbol{f}_{n+1} := f\big(y_n\boldsymbol{e} + h\delta(E)E^{-k+1}\boldsymbol{f}_n\big), \tag{3.1a}$$

where E denotes the forward shift operator, i.e., $E\boldsymbol{f}_n = \boldsymbol{f}_{n+1}$. The predictor method is now simply defined by

$$\boldsymbol{r}^{(0)} := \boldsymbol{f}_{n+1}. \tag{3.1b}$$

Here $\delta(\zeta)$ is a polynomial of degree $k-1$ whose coefficients are matrices of appropriate

dimension (cf. [7]). The method defined by (2.2) and (3.1) gives rise to a k-step PC method requiring $m+1$ right-hand side evaluations per step. For $m=0$, this method fits into the class of methods investigated in [7].

By Taylor expansion of $\boldsymbol{f}_{n+1}$ (or $\boldsymbol{Y}^{(0)}$), conditions for the satisfaction of $\boldsymbol{r}_{n+1}-\boldsymbol{f}_{n+1}=\mathrm{O}(h^q)$ can be derived in terms of A and $\delta(\zeta)$. For instance we have the following theorem.

Theorem 3. *Let the corrector defined by* $\{A, \boldsymbol{b}^T\}$ *be of order* p, *then the* k*-step PC method*

$$\begin{aligned}
&\boldsymbol{f}_{n+1}=f\big(y_n\boldsymbol{e}+h\delta(E)E^{-k+1}\boldsymbol{f}_n\big),\\
&\boldsymbol{r}^{(0)}=\boldsymbol{f}_{n+1}, \qquad \boldsymbol{r}^{(j)}=f\big(y_n\boldsymbol{e}+hA\boldsymbol{r}^{(j-1)}\big), \quad j=1,\ldots,m,\\
&y_{n+1}=y_n+h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(m)},
\end{aligned} \tag{3.2}$$

is of order $p^* := \min\{p, q+m\}$, *where*

$$\begin{aligned}
&q=2 \quad \textit{if} && A\boldsymbol{e}-\delta(1)\boldsymbol{e}=\boldsymbol{0}.\\
&q=3 \quad \textit{if, in addition,} && A^2\boldsymbol{e}-\delta^2(1)\boldsymbol{e}+k\delta(1)\boldsymbol{e}-\delta'(1)\boldsymbol{e}=\boldsymbol{0},\\
& && \tfrac{1}{2}A^2\boldsymbol{e}-\tfrac{1}{2}\delta^2(1)\boldsymbol{e}+k\delta(1)\boldsymbol{e}-\delta'(1)\boldsymbol{e}=\boldsymbol{0}.
\end{aligned}$$

Example 4. The most simple example is the case where $k=1$ and $\delta(\zeta)=0$, so that $\boldsymbol{r}^{(0)}=f(y_n)\boldsymbol{e}$ and $q=1$. This case has been already considered in the preceding section. Next we choose $k=1$ and $\delta(\zeta)=A$. It is readily verified that the order conditions for the predictor are satisfied for $q=2$. The algorithm (3.2) assumes the one-step form

$$\begin{aligned}
&\boldsymbol{f}_{n+1}=f\big(y_n\boldsymbol{e}+hA\boldsymbol{f}_n\big),\\
&\boldsymbol{r}^{(0)}=\boldsymbol{f}_{n+1}, \qquad \boldsymbol{r}^{(j)}=f\big(y_n\boldsymbol{e}+hA\boldsymbol{r}^{(j-1)}\big), \quad j=1,\ldots,m,\\
&y_{n+1}=y_n+h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(m)}.
\end{aligned} \tag{3.3}$$

If the RK corrector has order p, then by performing $m=p-2$ corrections this method is also of order p and requires $p-1$ right-hand side evaluations per step. Formally, the method no longer belongs to the class of one-step RK methods. However, in actual applications, the method is self-starting if we take $\boldsymbol{f}_0=f(y_0)\boldsymbol{e}$.

Finally, we choose $k=2$ and $\delta(\zeta)=2A\zeta-A$ which satisfy the order conditions for $q=3$. The algorithm (3.2) assumes the two-step form

$$\begin{aligned}
&\boldsymbol{f}_{n+1}=f\big(y_n\boldsymbol{e}+2hA\boldsymbol{f}_n-hA\boldsymbol{f}_{n-1}\big),\\
&\boldsymbol{r}^{(0)}=\boldsymbol{f}_{n+1}, \qquad \boldsymbol{r}^{(j)}=f\big(y_n\boldsymbol{e}+hA\boldsymbol{r}^{(j-1)}\big), \quad j=1,\ldots,m,\\
&y_{n+1}=y_n+h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(m)}.
\end{aligned} \tag{3.4}$$

If the RK corrector has order p, then by performing $m=p-3$ corrections this method is also of order p and requires $p-2$ right-hand side evaluations per step.

4. Stability

We consider linear stability with respect to the test equation

$$y'(t)=\lambda y(t). \tag{4.1}$$

It is easily verified that application of (2.5) yields the recursion

$$y_{n+1} = \left[1 + z\boldsymbol{b}^{\mathrm{T}}\boldsymbol{e} + z^2\boldsymbol{b}^{\mathrm{T}}A\boldsymbol{e} + z^3\boldsymbol{b}^{\mathrm{T}}A^2\boldsymbol{e} + \cdots + z^{m+1}\boldsymbol{b}^{\mathrm{T}}A^m\boldsymbol{e}\right] y_n, \tag{4.2}$$

where we have written $z = \lambda h$. The stability polynomial is given by

$$P_{m+1}(z) = 1 + z\boldsymbol{b}^{\mathrm{T}}\boldsymbol{e} + z^2\boldsymbol{b}^{\mathrm{T}}A\boldsymbol{e} + z^3\boldsymbol{b}^{\mathrm{T}}A^2\boldsymbol{e} + \cdots + z^{m+1}\boldsymbol{b}^{\mathrm{T}}A^m\boldsymbol{e}. \tag{4.3}$$

In the particular case where we choose $m = p - 1$, p being the order of the corrector, we obtain a stability polynomial of degree p. According to Theorem 1, this PIRK method is of order p so that the stability polynomial is consistent of order p, i.e., it approximates $\exp(z)$ with pth-order accuracy. Thus, we have proved the next theorem.

Theorem 5. *Let the corrector be of order p. If $m = p - 1$, then the method* (2.5) *becomes an* (*explicit*) *RK method with the stability polynomial*

$$P_p(z) = 1 + z + \frac{1}{2!}z^2 + \frac{1}{3!}z^3 + \cdots + \frac{1}{p!}z^p.$$

Using a result on truncated Taylor series (cf. [6,p.236]), we have the next corollary as a corollary of this theorem.

Corollary 6. *The method of Theorem* 5 *is stable in the interval* $[\beta_{\mathrm{real}}, 0]$, *where*

$$\beta_{\mathrm{real}} \approx 0.368\ (p+1)(19(p+1))^{1/(2(p+1))}. \tag{4.4}$$

Defining $[-\mathrm{i}\,\beta_{\mathrm{imag}}, \mathrm{i}\,\beta_{\mathrm{imag}}]$ to be the interval on the imaginary axis where the method of Theorem 5 is stable, we list in Table 2 the values of β_{real} (and its approximation provided by (4.4)) and β_{imag} for orders $p = 1, 2, \ldots, 10$.

5. Stepsize control

In this section we will describe a simple strategy to implement the aforementioned methods with a variable stepsize in order to control the local truncation error. This strategy is the same as the one employed by Hairer, Nørsett and Wanner [5,p.167] in their code DOPRI8, in which they have implemented the 13-stage, 8th-order explicit RK method with the embedded method of order 7 of Prince and Dormand.

This strategy is based on the observation that when iterating the equation (2.1b) for $\boldsymbol{r}_{n+1}$ we obtain approximations $\boldsymbol{r}^{(j)}$ of successively increasing order, i.e.,

$$\boldsymbol{r}^{(j)} - \boldsymbol{r}_{n+1} = \mathrm{O}(h^{\min\{p, q+j\}}), \quad j = 1, 2, \ldots, m.$$

Table 2
β_{real} and β_{imag} for the method of Theorem 5

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$	$p=6$	$p=7$	$p=8$	$p=9$	$p=10$
True value of β_{real}	2.00	2.00	2.52	2.78	3.22	3.55	3.95	4.31	4.70	5.07
Value according to (4.4)	1.83	2.17	2.53	2.90	3.28	3.65	4.03	4.41	4.78	5.16
True value of β_{imag}	0.00	0.00	1.73	2.82	0.00	0.00	1.76	3.39	0.00	0.00

Thus, apart from our final approximation $y_{n+1} := y_n + h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(m)}$, we can easily construct a reference solution (cf. (2.3))

$$y^{(k)} := y_n + h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(k)}, \tag{5.1}$$

for some $k < m$. Since $\boldsymbol{r}^{(k)}$ has already been computed, this does not require additional right-hand side evaluations. This reference solution $y^{(k)}$ can be considered as an "embedded" solution [5]. Now, as an estimate for the local error ϵ in the step from t_n to $t_{n+1} = t_n + h$, we take

$$\epsilon := \| y_{n+1} - y^{(k)} \|, \tag{5.2}$$

for some norm $\| \cdot \|$. Usually, one uses reference solutions $y^{(k)}$ such that the orders of y_{n+1} and $y^{(k)}$ differ by 1. Here we follow this approach and choose $k = m - 1$.

First, we will discuss the case where we restrict our stepsize strategy to methods in which the number of iterations m is fixed in each step and is given by $m = p - q$. Hence, $\boldsymbol{r}^{(m)} - \boldsymbol{r}_{n+1}$ and $\boldsymbol{r}^{(m-1)} - \boldsymbol{r}_{n+1}$ behave as $O(h^p)$ and $O(h^{p-1})$, respectively, and, consequently,

$$\epsilon = \| y_{n+1} - y^{(m-1)} \| = \| y_n + h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(m)} - y_n - h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(m-1)} \| = O(h^p).$$

Then ϵ is compared with some prescribed tolerance TOL and the step is accepted if $\epsilon \leqslant \mathrm{TOL}$, and rejected otherwise. Furthermore, the value of ϵ allows us to make an estimate for the asymptotically optimal stepsize:

$$h\left(\frac{\mathrm{TOL}}{\epsilon}\right)^{1/p},$$

which will be taken in the next step (or to recompute the current step in case of rejection). However, to give the code some robustness, we actually implemented (cf. [5,p.167])

$$h_{\mathrm{new}} = h \min\left\{6, \max\left\{\tfrac{1}{3}, 0.9\left(\frac{\mathrm{TOL}}{\epsilon}\right)^{1/p}\right\}\right\}. \tag{5.3}$$

The constants 6 and $\frac{1}{3}$ in this expression serve to prevent an abrupt change in the stepsize and the safety factor 0.9 is added to increase the probability that the next step will be accepted.

Apart from the variable stepsize implementation mentioned above, the PIRK methods allow for a simple extension of the control strategy by which also the *order* of the method may vary from step to step. This can be achieved by abandoning the approach of a fixed number of iterations. Referring to the description above, we can construct a *sequence* of reference solutions, i.e., after each iteration the "embedded" solution

$$y^{(j)} := y_n + h\boldsymbol{b}^{\mathrm{T}}\boldsymbol{r}^{(j)}$$

is computed. Then, we can use the difference of two successive reference solutions as an estimate for the local error, i.e.,

$$\epsilon^{(j)} := \| y^{(j)} - y^{(j-1)} \|.$$

If, during the iteration, the tolerance criterion $\epsilon^{(j)} \leqslant \mathrm{TOL}$ is satisfied for some $j = j_0 < m$, then there is no need to proceed with the iteration process and we accept $y^{(j_0)}$ as the numerical solution y_{n+1}. This suggests to try the next step with the value of m defined by $m = j_0$. Since

$$\epsilon^{(j_0)} = O(h^{p^*}), \quad p^* = \min\{p + 1, q + j_0\},$$

a prediction for the next stepsize can be made according to (5.3), where p is replaced by p^* and ϵ by $\epsilon^{(j_0)}$.

It may happen that the tolerance condition is *not* satisfied for $j = j_0 \leqslant m$. In such cases, the values of m and h predicted in the preceding step were not reliable. One may then decide to reject the current value of m and to continue the iteration process. This is particularly recommendable if the value of the current p^* is less than p. If the continuation of the iteration process does not help to satisfy the tolerance condition $\epsilon^{(j)} \leqslant \mathrm{TOL}$ for $j \leqslant M$, where M is some prescribed upper bound for the number of iterations per step, then the (relatively costly) alternative is rejection of the current value of h, to redefine h according to (5.3) using the most recent information on the error, and to perform the present step once again. In this way a variable order variable stepsize RK method can be constructed.

6. Numerical experiments

We present few examples illustrating the efficiency of PIRK methods on parallel computers. The calculations are performed using 14-digits arithmetic. The methods tested were all applied in P(EC)^{m}E mode.

6.1. Comparison of various predictor methods

In order to examine the effect of various predictor methods on the efficiency of the PIRK algorithm we performed a few tests by integrating the equation of motion for a rigid body without external forces (cf. [8, Problem B5]):

$$\begin{aligned} y_1' &= y_2 y_3, & y_1(0) &= 0, \\ y_2' &= -y_1 y_3, & y_2(0) &= 1, \\ y_3' &= -0.51\, y_1 y_2, & y_3(0) &= 1, \quad 0 \leqslant t \leqslant T. \end{aligned} \tag{6.1}$$

In these tests we used the 10th-order Gauss–Legendre corrector and the following predictor methods:

Predictor I:	$\boldsymbol{r}^{(0)} = f(y_n)\boldsymbol{e}$ (cf. (2.5)),	$q = 1$,	$p = \min\{m+1, 10\}$,
Predictor II:	$\boldsymbol{r}^{(0)}$ defined by standard 4th-order RK,	$q = 5$,	$p = \min\{m+5, 10\}$,
Predictor III:	$\boldsymbol{r}^{(0)} = f(y_n\boldsymbol{e} + hA\boldsymbol{f}_n)$ (cf. (3.3)),	$q = 2$,	$p = \min\{m+2, 10\}$,
Predictor IV:	$\boldsymbol{r}^{(0)} = f(y_n\boldsymbol{e} + 2hA\boldsymbol{f}_n - hA\boldsymbol{f}_{n-1})$ (cf. (3.4)),	$q = 3$,	$p = \min\{m+3, 10\}$.

In Table 3 we have listed the values $D \backslash N$, where D denotes the number of correct decimal digits at the endpoint, i.e., we write the maximum norm of the error at $t = T$ in the form 10^{-D}, and where N denotes the total number of effective right-hand side evaluations performed during the integration process. Furthermore, we indicated the effective order p_{eff}, that is the order of accuracy which is shown numerically.

Comparing experiments with equal N (notice that this table contains for each h and each predictor an experiment with $N = 180h^{-1}$) we conclude that in most experiments the 3rd-order

Table 3
Values $D\backslash N$ for problem (6.1) with $T = 20$

h^{-1}	Predictor I			Predictor II			Predictor III			Predictor IV	
	$m = 8$	$m = 9$	$m = 10$	$m = 4$	$m = 5$	$m = 6$	$m = 7$	$m = 8$	$m = 9$	$m = 7$	$m = 8$
1	5.6\180	6.5\200	6.9\220	5.3\180	7.0\200	6.8\220	4.8\160	5.5\180	7.5\200	4.6\160	5.7\180
2	8.0\360	9.7\400	9.8\440	7.8\360	10.2\400	9.7\440	7.2\320	8.5\360	9.6\400	7.2\320	8.8\360
4	10.6\720	13.0\800	12.3\880	10.5\720	13.3\800	12.2\880	9.7\640	11.6\720	12.1\800	10.4\640	12.4\720
$p_{\text{eff}} \approx$	9	10	10	9	10	10	9	10	10	10	10

predictor IV and the 2nd-order predictor III yield the most accurate values. However, the price we pay is more storage and a more complicated implementation because of the auxiliary recursion for f_n. The predictors I and II produce comparable accuracies. As the added storage for the predictors III and IV is not offset by comparable reduction in the volume of computation, we recommend predictor I in actual computations. The resulting PIRK method is a true one-step RK method of an extremely simple structure, and consequently allowing an easy and straightforward implementation. A FORTRAN code based on this PIRK method can be found in the Appendix to this paper.

6.2. Comparison with the 10th-order methods of Curtis and Hairer

Curtis [2] and Hairer [4] used the test problem (6.1) for testing and comparing their 10th-order RK methods. In Table 4 the results of the experiments performed by Curtis and Hairer are reproduced together with results obtained by the PC pairs consisting of the predictors I, II and III, and the 10th-order Gauss–Legendre corrector. Again we see that the simple predictor I can compete favourable with the predictors II and III.

6.3. Comparison with the 8(7)-method of Prince and Dormand

The 8(7)-method of Prince and Dormand [13] is nowadays generally considered as one of the most efficient methods with automatic stepsize control for TOL-values approximately in the

Table 4
Values $D\backslash N$ for problem (6.1) with $T = 60$

Method	p	$60/h$	D	N
Runge–Kutta	4	12000	9.6	48000
Adams–Moulton–Bashforth	4	6000	8.1	12000
Bulirsch–Stoer: polynomial extrapolation	–	–	8.9	5276
Bulirsch–Stoer: rational extrapolation	–	–	9.6	4860
Runge–Kutta–Curtis	10	240	9.9	4320
Runge–Kutta–Hairer	10	240	10.1	4080
(2.2) with predictor I and $m = 9$	10	156	10.0	1560
(2.2) with predictor I and $m = 10$	10	150	10.0	1650
(2.2) with predictor II and $m = 5$	10	150	10.1	1500
(2.2) with predictor II and $m = 6$	10	156	10.1	1716
(2.2) with predictor III and $m = 8$	10	210	10.0	1891
(2.2) with predictor III and $m = 9$	10	168	10.0	1681

Table 5
Values of N for problem (6.2)

Method	$D=5$	$D=6$	$D=7$	$D=8$	$D=9$	$D=10$	$D=11$
DOPRI8	595	759	963	1227	1574	1990	2503
PIRK8	379	495	623	786	978	1383	1874
PIRK10	327	388	490	704	884	977	1078

range 10^{-7} to 10^{-13}. In this subsection we compare the DOPRI8 code, as given by [5], with the PIRK method based on predictor I and the Gauss–Legendre correctors of orders 8 and 10. To let the comparison of the DOPRI8 code and the PIRK codes not be influenced by a different stepsize strategy, we equipped the PIRK codes with the same strategy (see Section 5). These codes are respectively denoted by PIRK8 and PIRK10.

6.3.1. Fehlberg problem

As a first test problem we take an example from [3]:

$$\begin{aligned} y_1' &= 2ty_1 \log\bigl(\max\{y_2, 10^{-3}\}\bigr), \qquad y_1(0)=1, \\ y_2' &= -2ty_2 \log\bigl(\max\{y_1, 10^{-3}\}\bigr), \qquad y_2(0)=\mathrm{e}, \end{aligned} \qquad 0 \leqslant t \leqslant 5, \tag{6.2}$$

with exact solution $y_1(t)=\exp(\sin(t^2))$, $y_2(t)=\exp(\cos(t^2))$. For tolerances TOL running from 10^{-5} up to 10^{-12} we computed the D and corresponding $\log_{10}(N)$ values. Instead of presenting the polygon graph for these values as was done in [5], we preferred to present the $D\backslash N$ lying on this polygon for a number of integer values of D. In Table 5 these values are listed.

6.3.2. Euler equations

Next, we apply the codes to Euler's equation for a rigid body (cf. (6.1)). The performance of the code is presented in Table 6.

6.3.3. Orbit equations

Finally, we apply the codes to the orbit equations (cf. [8, Problem D2]):

$$\begin{aligned} y_1' &= y_3, & y_1(0) &= 1-\epsilon, \\ y_2' &= y_4, & y_2(0) &= 0, \\ y_3' &= \frac{-y_1}{\bigl(y_1^2+y_2^2\bigr)^{3/2}}, & y_3(0) &= 0, \\ y_4' &= \frac{-y_2}{\bigl(y_1^2+y_2^2\bigr)^{3/2}}, & y_4(0) &= \sqrt{\frac{1+\epsilon}{1-\epsilon}}, \quad \epsilon=\tfrac{3}{10}, \quad 0 \leqslant t \leqslant 20. \end{aligned} \tag{6.3}$$

Table 6
Values of N for problem (6.1)

Method	$D=6$	$D=7$	$D=8$	$D=9$	$D=10$	$D=11$	$D=12$
DOPRI8	415	576	728	898	1133	1422	1817
PIRK8	294	381	534	728	961	1172	1746
PIRK10	252	297	357	426	580	730	920

Table 7
Values of N for problem (6.3)

Method	$D=5$	$D=6$	$D=7$	$D=8$	$D=9$	$D=10$	$D=11$
DOPRI8	615	723	831	1062	1284	1780	2024
PIRK8	463	559	679	859	1099	1411	1876
PIRK10	378	448	540	662	784	911	1076

The performance of the codes is presented in Table 7.

An obvious conclusion which can be drawn, is that—at least for these three testexamples—both PIRK codes are more efficient than DOPRI8; in the average, PIRK8 requires $\frac{3}{4}$ of the number of f-evaluations that are needed by DOPRI8 to yield the same accuracy, whereas PIRK10 is almost twice as efficient. The superiority of PIRK10, especially in the high-accuracy range, is undoubtedly due to its higher order. Therefore, it would be interesting to compare this method with an embedded (sequential) Runge–Kutta pair of comparable order. Unfortunately, to the best of our knowledge, such formulae have not been constructed in the literature.

7. Conclusions

Iterated Runge–Kutta methods of arbitrarily high order have been constructed that are capable of efficiently exploiting the parallelism of an MIMD computer architecture. Assuming that sufficient processors are available, it is shown how to derive "optimal methods", i.e., methods requiring a number of parallelised f-evaluations equal to the order. Within the class of optimal methods considered, the required number of processors s is least with respect to the order p if the algorithm is based on an iterated Gauss–Legendre RK method and this minimal number is given by $s = \frac{1}{2}p$. It is known that optimal methods exist requiring a smaller number of processors (an example is the 5th-order method of Butcher, mentioned in the Introduction), but it is not clear how to formulate a general construction procedure to arrive at such methods for arbitrary order.

A nice feature of the methods proposed is that they provide an embedded reference solution without additional f-evaluations. This advantage has been utilized to make a variable step implementation which has been compared with the code DOPRI8, nowadays considered as "the state of the art" for the automatic integration of ODEs. On the basis of some testexamples, the performance of the new code is compared with DOPRI8 and, in terms of the required number of f-evaluations, demonstrates a superior behaviour.

Another aspect is the simple implementation of the new algorithm. In the Appendix a FORTRAN subroutine is provided which accepts a general RK method of arbitrary order, defined in terms of its Butcher tableau. For example, if there is need for an automatic integration routine of order higher than 8, as is furnished by DOPRI8, then we can suffice to specify, e.g., a high-order Gauss method (the construction of which is simple and fully described in [1]) and call this subroutine. Furthermore, for such accuracy demands, we remark that even in the case that the parallel evaluation of the derivatives is not possible (e.g., on a uniprocessor machine) or not relevant (e.g., because the evaluation of f is very inexpensive and offset by the overhead), this code may still be of value. Since classical embedded RK pairs of such high orders are lacking, it

may turn out that, even in the nonparallelised form, the present code is more efficient than DOPRI8, in spite of its large redundancy with respect to the number of f-evaluations (cf. the discussion following Theorem 1). It is easily verified that this approach can offer sequential embedded RK methods of arbitrary order p, using $ms + 1 = \frac{1}{2}(p^2 - p + 2)$ stages. This aspect, which is a direct consequence of the simplicity of the PIRK algorithm, needs further investigation.

Appendix

Here we give the implementation (in FORTRAN 77) of the optimal PIRK methods of the form (2.5), including error control. This subroutine offers the user the facility to specify an arbitrary Runge–Kutta method by means of the matrix A and the vectors $\boldsymbol{b}^T$ and $\boldsymbol{c}$ (see also the description of these parameters).

Although this routine has been coded in standard FORTRAN 77, it will require machine-dependent amendment as to exploit the parallelism. Therefore we shall discuss in some detail the most important loop in this subroutine, i.e., the 80-loop. It is here, that the parallel calculation of the components of the iterate $\boldsymbol{r}^{(j)}$ is to be performed (cf. (2.2)). A first observation is that this loop contains a call to another subprogram (viz. FCN). The separate compilation of subprograms prevents the compiler from actually parallelising this loop, since it is unknown what happens within FCN. Nevertheless, if the present source is offered to a compiler without giving any instructions, the outcome (i.e., the "optimized" object code) will be the product of all kinds of operations, like unravelling, interchanging, distributing loops etc., and will certainly speedup the execution. However, the parallelisation will probably not completely fit in with the ideas as advocated in the present paper. Therefore, we have to insert an explicit specification concerning the way the compiler had to do its job; for example, we can specify that it is in this case without any danger to parallelise over the FCN-calls. Most parallel computers offer so-called "directives" for this purpose (e.g., using an Alliant, one can specify: cvd$ cncall). Since these directives may differ for the various parallel machines, we decided to code this loop in standard FORTRAN.

Another observation is that the 80-loop contains two nested innerloops: one over the components of the ODE and one to form the innerproduct of a row of A and the iterate vector $\boldsymbol{r}^{(j-1)}$. If the parallel machine at hand has an architecture in which each processor is a vectorprocessor, then it may be advantageous to interchange these innerloops. Such considerations depend on the dimension of the ODE, the startup time of the particular vectorprocessor, the "smartness" of the compiler, etc.

To sum up, in order to obtain an optimal performance, the user of the subroutine PIRK is advised to adjust the 80-loop to the specific situation he is dealing with, like the number of processors available (perhaps even larger than s), the dimensions of the problems to be solved, etc.

```
      SUBROUTINE PIRK(N, NR, FCN, T, Y, TEND, TOL, H, S, P,
     +                NRA, A, B, C, YN, FN, RJ, RJM1, BIGY, YREF)
C-----------------------------------------------------------------
C   PIRK SOLVES AN INITIAL VALUE PROBLEM FOR A SYSTEM OF FIRST-ORDER
C   DIFFERENTIAL EQUATIONS OF THE FORM Y'(T)=F(T,Y(T)).
C   THE ROUTINE IS BASED ON AN ITERATED RUNGE-KUTTA METHOD AND
C   DESIGNED IN SUCH A WAY THAT PARALLELISM IS EXPLOITED.
C   IN COUNTING THE NUMBER OF REQUIRED F-EVALUATIONS, IT IS ASSUMED
C   THAT THE NUMBER OF STAGES IN THE RUNGE-KUTTA METHOD DOES NOT
```

```
C   EXCEED THE NUMBER OF PROCESSORS AVAILABLE.
C
C   MEANING OF THE PARAMETERS:
C   --------------------------
C   N      - INTEGER VARIABLE
C            THE DIMENSION OF THE SYSTEM
C   NR     - INTEGER VARIABLE
C            FIRST DIMENSION OF THE ARRAYS RJ, RJM1 AND BIGY AS
C            DECLARED IN THE CALLING PROGRAM  (NR .GE. N)
C   FCN    - SUBROUTINE
C            A USER-DEFINED SUBROUTINE COMPUTING THE DERIVATIVE
C            F(T,Y(T))
C            ITS SPECIFICATION READS:
C               SUBROUTINE FCN(N,T,Y,F)
C               DIMENSION Y(N),F(N)
C
C               ON RETURN, F(I) (I=1,...,N) MUST CONTAIN THE VALUE OF
C               THE I-TH COMPONENT OF THE DERIVATIVE VECTOR
C               FCN MUST BE DECLARED EXTERNAL IN THE CALLING PROGRAM
C   T      - REAL VARIABLE
C            THE INDEPENDENT VARIABLE; ON ENTRY, T SHOULD BE SET
C            TO THE INITIAL VALUE. ON RETURN, T CONTAINS THE VALUE
C            FOR WHICH Y IS THE SOLUTION
C   Y      - REAL ARRAY OF DIMENSION (AT LEAST) N
C            THE DEPENDENT VARIABLE. ON ENTRY, Y SHOULD CONTAIN THE
C            INITIAL VALUES OF THE DEPENDENT VARIABLES.
C            ON RETURN, Y CONTAINS THE NUMERICAL SOLUTION AT T
C   TEND   - REAL VARIABLE
C            TEND SPECIFIES THE END POINT OF THE RANGE OF INTEGRATION
C   TOL    - REAL VARIABLE
C            TOL (>0) SPECIFIES A BOUND FOR THE LOCAL TRUNCATION
C            ERROR
C   H      - REAL VARIABLE
C            ON ENTRY, H SHOULD BE GIVEN A VALUE WHICH IS USED AS A
C            GUESS FOR THE INITIAL STEP SIZE
C   S      - INTEGER VARIABLE
C            NUMBER OF STAGES OF THE SPECIFIED RUNGE-KUTTA METHOD
C   P      - INTEGER VARIABLE
C            ORDER OF ACCURACY OF THE SPECIFIED RUNGE-KUTTA METHOD
C   NRA    - INTEGER VARIABLE
C            FIRST DIMENSION OF THE ARRAY A AS DECLARED IN THE
C            CALLING PROGRAM  (NRA .GE. S)
C   A      - REAL ARRAY OF DIMENSION (NRA,L) WITH L .GE. S
C   B      - REAL ARRAY OF DIMENSION (AT LEAST) S
C   C      - REAL ARRAY OF DIMENSION (AT LEAST) S
C
C            THE PARAMETERS A, B AND C DEFINE THE RUNGE-KUTTA METHOD,
C            WRITTEN IN THE SO-CALLED BUTCHER-NOTATION (USUALLY, THE
C            ELEMENTS OF C ARE EQUAL TO THE ROW-SUMS OF THE MATRIX A)
C            IN PRINCIPLE, ANY RUNGE-KUTTA METHOD CAN BE USED.
C            HOWEVER, THE OPTIMAL ORDER WITH RESPECT TO THE NUMBER OF
C            STAGES IS OBTAINED IF A 'GAUSS-LEGENDRE' METHOD IS
C            SELECTED. THE CORRESPONDING A, B AND C CAN BE FOUND IN:
C            J.C. BUTCHER, IMPLICIT RUNGE-KUTTA PROCESSES, MATH.COMP.
C                          18,(1964) PP. 50-64
C   YN     - REAL ARRAY OF DIMENSION (AT LEAST) N
C            USED AS SCRATCH ARRAY
C   FN     - REAL ARRAY OF DIMENSION (AT LEAST) N
C            USED AS SCRATCH ARRAY
C   RJ     - REAL ARRAY OF DIMENSION (NR,L) WITH L .GE. S
C            USED AS SCRATCH ARRAY
C   RJM1   - REAL ARRAY OF DIMENSION (NR,L) WITH L .GE. S
C            USED AS SCRATCH ARRAY
C   BIGY   - REAL ARRAY OF DIMENSION (NR,L) WITH L .GE. S
C            USED AS SCRATCH ARRAY
C   YREF   - REAL ARRAY OF DIMENSION (AT LEAST) N
C            USED AS SCRATCH ARRAY
```

```
C-----------------------------------------------------------------------
      DIMENSION Y(N),YN(N),FN(N),YREF(N),RJ(NR,*),RJM1(NR,*),
     +          BIGY(NR,*),A(NRA,*),B(*),C(*)
      INTEGER S,P
      LOGICAL REJECT
C-----------------------------------------------------------------------
C   THE COMMON BLOCK STAT CAN BE USED FOR STATISTICS CONCERNING THE
C   INTEGRATION PROCESS
C       NFCN      NUMBER OF EVALUATIONS OF THE DERIVATIVE FUNCTION F
C       NSTEPS    NUMBER OF INTEGRATION STEPS
C       NACCPT    NUMBER OF ACCEPTED STEPS
C       NREJCT    NUMBER OF REJECTED STEPS
C-----------------------------------------------------------------------
      COMMON/STAT/NFCN,NSTEPS,NACCPT,NREJCT
C-------------------------------------------------------
C   SMALLEST NUMBER SATISFYING 1.0 + UROUND > 1.0
C   UROUND MAY REQUIRE AMENDMENT ON DIFFERENT MACHINES
C-------------------------------------------------------
      DATA UROUND/7.1E-15/
C------------------
C   INITIALISATIONS
C------------------
      REJECT=.FALSE.
      NFCN=0
      NSTEPS=0
      NACCPT=0
      NREJCT=0
      TOL=AMAX1(TOL,10.0*UROUND)
C-------------------------------------------------------------
C   ON ITERATING THE RUNGE-KUTTA METHOD, WE USE A PREDICTION
C   OF FIRST-ORDER. THEREFORE, WE NEED M=P-1 ITERATIONS TO
C   OBTAIN A RESULT OF ORDER P.
C-------------------------------------------------------------
      M=P-1
C-------------------
C   INTEGRATION STEP
C-------------------
   10 CONTINUE
      IF(H .LT. 10.0*UROUND)THEN
         WRITE(6,1)T
    1    FORMAT('   THE ROUTINE HAS ADVANCED THE SOLUTION UP TO T=',
     +          E16.8,/,'  AND STOPPED BECAUSE THE STEP SIZE HAS',
     +          ' BECOME TOO SMALL'/'   TRY A LESS STRINGENT VALUE',
     +          ' OF TOL OR CHANGE TO A HIGHER-ORDER METHOD')
         RETURN
      ENDIF
      IF(TEND-T .LT. UROUND)RETURN
      IF(T+H .GT. TEND)H=TEND-T
C----------------------
C   FORM THE PREDICTION
C----------------------
      DO 20 I=1,N
   20    YN(I)=Y(I)

      CALL FCN(N,T,YN,FN)
      NFCN=NFCN+1
   30 NSTEPS=NSTEPS+1
      DO 50 L=1,S
         DO 40 I=1,N
   40       RJM1(I,L)=FN(I)
   50    CONTINUE
C---------------------------------------------
C   IN THE 110-LOOP, THE ITERATION IS PERFORMED
C---------------------------------------------
      DO 110 J=1,M
C-----------------------------------------------------------
C   IN THE 80-LOOP, THE S STAGES ARE PERFORMED CONCURRENTLY
C-----------------------------------------------------------
         DO 80 L=1,S
            DO 70 I=1,N
```

```
               BIGY(I,L)=YN(I)
               DO 60 K=1,S
   60             BIGY(I,L)=BIGY(I,L)+H*A(L,K)*RJM1(I,K)
   70          CONTINUE
            CALL FCN(N,T+C(L)*H,BIGY(1,L),RJ(1,L))
   80       CONTINUE
         NFCN=NFCN+1
C---------------------
C   SHIFT THE ITERATES
C---------------------
         IF(J.LT.M)THEN
            DO 100 L=1,S
               DO 90 I=1,N
   90             RJM1(I,L)=RJ(I,L)
  100          CONTINUE
         ENDIF
  110    CONTINUE
C-------------------------------------------
C   CALCULATE THE FINAL SOLUTION OF THIS STEP
C   AND A REFERENCE SOLUTION FOR ERROR CONTROL
C-------------------------------------------
      DO 130 I=1,N
         Y(I)=YN(I)
         YREF(I)=YN(I)
         DO 120 K=1,S
            Y(I)=Y(I)+H*B(K)*RJ(I,K)
  120       YREF(I)=YREF(I)+H*B(K)*RJM1(I,K)
  130    CONTINUE
C---------------
C   ERROR CONTROL
C---------------
      ERROR=0.0
      DO 140 I=1,N
         DENOM=AMAX1(1.0E-6, ABS(Y(I)), ABS(YN(I)), 2.0*UROUND/TOL)
  140    ERROR=ERROR+((Y(I)-YREF(I))/DENOM)**2
      ERROR=SQRT(ERROR/N)
      FAC=AMAX1(1.0/6.0,AMIN1(3.0,(ERROR/TOL)**(1.0/P)/0.9))
      HNEW=H/FAC
      IF(ERROR.GT.TOL)THEN
C------------------
C   STEP IS REJECTED
C------------------
         IF(NACCPT.GE.1)NREJCT=NREJCT+1
         REJECT=.TRUE.
         H=HNEW
         GOTO 30
      ELSE
C------------------
C   STEP IS ACCEPTED
C------------------
         NACCPT=NACCPT+1
         T=T+H
         IF(REJECT)THEN
            HNEW=AMIN1(HNEW,H)
            REJECT=.FALSE.
         ENDIF
         H=HNEW
         GOTO 10
      ENDIF
      END
```

References

[1] J.C. Butcher, Implicit Runge–Kutta processes, *Math. Comp.* **18** (1964) 50–64.

[2] A.R. Curtis, High-order explicit Runge–Kutta formulae, their uses, and limitations, *J. Inst. Math. Appl.* **16** (1975) 35–55.

[3] E. Fehlberg, Classical fifth-, sixth-, seventh-, and eighth-order Runge–Kutta formulas with stepsize control, NASA Technical Report **287**, 1968; extract published in *Computing* **4** (1969) 93–106.
[4] E. Hairer, A Runge–Kutta method of order 10, *J. Inst. Math. Appl.* **21** (1978) 47–59.
[5] E. Hairer, S.P. Nørsett and G. Wanner, *Solving Ordinary Differential Equations I. Nonstiff Problems* (Springer, Berlin, 1987).
[6] P.J. van der Houwen, *Construction of Integration Formulas for Initial Value Problems* (North-Holland, Amsterdam, 1977).
[7] P.J. van der Houwen, B.P. Sommeijer and P.A. van Mourik, Note on explicit parallel multistep Runge–Kutta methods, *J. Comput. Appl. Math.* **27** (3) (1989) 411–420.
[8] T.E. Hull, W.H. Enright, B.M. Fellen and A.E. Sedgwick, Comparing numerical methods for ordinary differential equations, *SIAM J. Numer. Anal.* **9** (1972) 603–637.
[9] A. Iserles and S.P. Nørsett, On the theory of parallel Runge–Kutta methods, Report DAMTP **1988/NA12**, University of Cambridge, 1988.
[10] K. Jackson and S.P. Nørsett, Parallel Runge–Kutta methods, 1988; to appear.
[11] I. Lie, Some aspects of parallel Runge–Kutta methods, Report No. **3/87**, University of Trondheim, Division Numerical Mathematics, 1988.
[12] S.P. Nørsett and H.H. Simonsen, Aspects of parallel Runge–Kutta methods, in: A. Bellen, Ed., *Workshop on Numerical Methods for Ordinary Differential Equations, L'Aquila, 1987,* Lecture Notes in Mathematics (Springer, Berlin, 1989).
[13] P.J. Prince and J.R. Dormand, High order embedded Runge–Kutta formulae, *J. Comput. Appl. Math.* **7** (1981) 67–75.